

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**Authoring Arbitrary XML Documents using DHTML  
and XSLT**

Inventor(s):  
Adriana Ardeleanu  
Jean Paoli  
Steve Mooney

ATTORNEY'S DOCKET NO. ms1-583

00599813-062100

## RELATED APPLICATIONS

The following patent applications are related to the present application, are assigned to the assignee of this patent application, and are expressly incorporated by reference herein:

- U.S. Patent Application Serial No. \_\_\_\_\_, entitled "Single Window Navigation Methods and Systems", bearing attorney docket number MS1-560us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. \_\_\_\_\_, entitled "Methods and Systems of Providing Information to Computer Users", bearing attorney docket number MS1-557us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. \_\_\_\_\_, entitled "Methods, Systems, Architectures and Data Structures For Delivering Software via a Network", bearing attorney docket number MS1-559us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. \_\_\_\_\_, entitled "Network-based Software Extensions", bearing attorney docket number MS1-563us, and filed on the same date as this patent application;
- U.S. Patent Application Serial No. \_\_\_\_\_, entitled "Architectures For And Methods Of Providing Network-based Software Extensions", bearing attorney docket number MS1-586us, and filed on the same date as this patent application.
- U.S. Patent Application Serial No. \_\_\_\_\_, entitled "Task Sensitive Methods And Systems For Displaying Command Sets", bearing attorney docket number MS1-562us, and filed on the same date as this patent application.

## TECHNICAL FIELD

This invention relates to authoring extensible markup language (XML) documents using dynamic hypertext markup language (DHTML)

## BACKGROUND

Extensible markup language (XML) is increasingly becoming the preferred format for transferring data. XML is a tag-based hierarchical language that is extremely rich in terms of the data that it can be used to represent. For example, XML can be used to represent data spanning the spectrum from semi-structured data (such as one would find in a word processing document) to generally structured data (such as that which is contained in a table). XML is well-suited for many types of communication including business-to-business and client-to-server communication.

Given the breadth of data that can be represented by XML, challenges arise when one wishes to provide a user interface to the data that a user can use to manipulate the data or the structure that contains the data. The classical approach to the user interface problem, outside of the XML environment, has been to use different UI technologies for different types of data (e.g. document, tabular data). This approach is clearly not the best when, with XML, it is more likely that a user will encounter and wish to interact with data that is both structured and unstructured. There have been some attempts at solving the problem of enabling a user to manipulate an XML document, but to date, they are extremely inflexible and do not appreciate the full power behind XML and XSL-T, the latter being a transformation that could be used to transform XML into Dynamic HTML or DHTML. For more information on XML, XSLT and XSD, the reader is referred to the following documents which are the work of, and available from the W3C (World Wide Web consortium): XML Schema Part 0: Primer, Extensible Markup Language (XML) 1.0, XML Schema Part 1: Structures, and XSL Transformations (XSLT) Version 1.0.



1 DHTML presentation. One manifestation of XSL-T is that the resultant DHTML  
2 structure may bear little resemblance to the corresponding XML tree structure that  
3 contains the data that is used by the XSLT to provide the DHTML.

4 The transition from XML to DHTML is then accomplished through XSL-T.  
5 This is generally a one way transition in which data that is described in XML is  
6 transformed into a presentation format for the user. Preserving the user experience  
7 of being able to interact with the data through its presentation format (e.g.  
8 DHTML) is crucial. While the transformation from XML to DHTML is fairly  
9 straightforward, there has been no clear transformation that would be the inverse  
10 of this transformation (i.e. transforming DHTML to XML) in a manner that is  
11 flexible and appreciates the full power of XSL-T. That is, while there are simple  
12 solutions to this problem, the robust nature of XSL-T and the differences in the  
13 corresponding XML and DHTML trees make it extremely difficult to attempt  
14 inverse transformation solutions.

15 There are solutions that enable a user to enter data in a DHTML document  
16 which is then copied back to the XML document. These solutions do not,  
17 however, enable a user to change the *structure* of the XML tree that represents the  
18 data. Additionally, there are solutions that are hardcoded solutions that can enable  
19 some manipulation of the XML tree given a DHTML modification, but the  
20 hardcoded nature of the solutions make them very specific to the data and XML  
21 tags with which they are used. For example, one of the XSL-T templates might  
22 include a hardcoded solution that allows a user to make structural changes to a  
23 table, such as adding a new row. This hardcoded solution is then only usable in  
24 connection with the table for which it was specifically defined. If a developer  
25 wishes to use the hardcoded solution for a different table, they must physically



In addition, the notion of a crystal is introduced and is used to map changes in a DHTML view directly back to a corresponding XML document. A crystal, in a basic form, includes one or more behaviors and associated XSL-T. In the illustrated example, a behavior is implemented as binary code that is associated with or attached to DHTML tags that are generated by the XSL-T. The crystals are used to transform XML into the DHTML views. The behaviors of a crystal are defined to be data-shape specific or dependent, with the data shape being defined by the XML document. The behavior is not necessarily dependent upon any schema, data or tags. Because of its data-shape dependent nature, crystals can be packaged for reuse with various XML documents which have no relation to one another other than a shape that is defined by the XML.

Behaviors can be attached to DHTML tags that are generated by the XSL-T. The behaviors ensure that user interactions with the DHTML view are mapped directly back to the XML document. In this way, the XML document can be authored to reflect the changes that are made to the DHTML view by the user.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

Fig. 1 is a high level block diagram that illustrates an XML document, an XSLT transformation, a DHTML view and a XSD or schema.

Fig. 2 is a high level diagram of a computer system that can be utilized to implement the described embodiments.

Fig. 3 is a flow diagram that describes steps in a method in accordance with one described embodiment.

Fig. 4 is a block diagram that illustrates one aspect of how changes to a DHTML view get mapped back to a corresponding XML document.

1 Fig. 5 is a flow diagram that describes steps in a method in accordance with  
2 one described embodiment.

### 3 4 **DETAILED DESCRIPTION**

#### 5 **Overview**

6 Methods and systems of authoring XML using DHTML views are  
7 described. In one implementation, various user interfaces can be automatically or  
8 semi-automatically provided in a DHTML view that then enable a user to interact  
9 with a DHTML view and change values (e.g. text or properties) of an associated  
10 DHTML tree. Value changes are translated to modifications of an associated  
11 XML structure. A transformation, e.g. an XSL-T, is applied to the modified XML  
12 structure which then changes the DHTML view to reflect the user's interaction.  
13 The interfaces, some of which are termed "in document" interfaces, permit a user  
14 to interact with a DHTML view and have those interactions reflected in a  
15 corresponding XML document that describes data that is associated with the  
16 DHTML view. These modifications can be made regardless of the complexity of  
17 the XSL-T that was utilized to transform the XML into the DHTML. Presentation  
18 of the various in document interfaces takes place by considering not only an XML  
19 schema (of which the XML document is an instance), but an XSL-T (XSLT  
20 transformation) that was utilized to transform the XML document into the  
21 DHTML view.

22 In another implementation, the notion of a crystal is introduced. A crystal,  
23 in a basic form, includes one or more behaviors and associated XSL-T. The  
24 crystals are used to transform XML into the DHTML views. The behaviors of a  
25 crystal are defined to be data-shape specific or dependent, with the data shape



being defined by the XML document. The behavior is not necessarily dependent upon any schema, data or tags. Because of its data-shape dependent nature, crystals can be packaged for reuse with various XML documents which have no relation to one another other than a shape that is defined by the XML. In the described implementation, behaviors are attached to the DHTML tags that are generated by the XSL-T. The behaviors ensure that user interactions with the DHTML view are mapped directly back to the XML document. In this way, the XML document can be authored to reflect the changes that are made to the DHTML view by the user. Because crystals are data shape-dependent and not schema dependent, as the shape is defined by the XML document, they can be used for authoring fragments of XML belonging to different schemas; those fragments simply share the same shape.

In this document, the following terminology will be used:

- **Schema** - a file (e.g. an XSD file) describing the schema for a particular type of XML document; schemas typically contain predefined tags and attributes that define the shape of the XML trees that represent an XML document; the schema provides a structure that each XML document must comply with; while editing an XML document, the schema is accessible through an instantiated DOM (document object model) (XDR DOM). Alternately, relevant information can be obtained from the schema and cached for use.
- **XML document** - an instance of an XML schema. Theoretically, for one schema there could be an infinite number of documents that instantiate the schema. When editing a document, the initial version and the final version of the document both adhere to the same schema, though the documents themselves are different. While processing, the XML document is instantiated through an instantiated DOM (XML DOM).

**XSLT transformation** – an XML file that transforms the XML document into an HTML view; for each XML document there could be any number of XSLT transformations, each creating a different HTML view over the

same document. An XSL-T file consists of one or more templates that match elements in the XML document. The XSL-T file that is initially authored by the application author is transformed by NetDocs when applied in edit mode into a NetDocs editing aware XSL-T. This transformation may break out templates into multiple templates, and add the appropriate behaviors (see below) based on NetDocs-specific hints added by the application developer. While editing the XML document, the transformed XSL-T is accessible to NetDocs through an instantiated DOM (XSL-T DOM).

- **DHTML view** – this is the result of the XSLT transformation applied on the XML document. The DHTML tree contains visual cues for displaying the data, but also behaviors. These behaviors are introduced by the XSLT transformation. While there could be behaviors introduced by the author of the XSLT transformation, there are behaviors introduced by NetDocs when it applies the transformation. These latter behaviors hold the logic for:
  - Copying to the XML DOM the values of the HTML leaf nodes that are modified
  - Determining, based on the cursor location in the HTML document, what editing services are available in the editing context. The editing context is determined by the HTML context in conjunction with the XSD context and the XSL-T template that was applied to generate that part of the view. The service is made known to the user
    - In-place (in the editing area) for pre-defined UI structures (e.g. table, grid, calendar control, label)
    - Enabling the appropriate XML editing context blocks in the NetDocs ContextBlock area.
  - Modifying the structure of the XML DOM based on the editing service selected
  - Incrementally updating the HTML view, by refreshing just the part of the view that is affected by the changes to the XML DOM.

### Exemplary Computing Environment

The embodiment described just below can be employed in connection with various computer systems. A computer system, for purposes of this document, can be considered as any computing device that includes some type of processor, i.e. a microprocessor, and some type of operating system. Thus, a computer

1 system can be construed to include, without limitation, traditional desktop  
2 computers, more powerful servers, various hand-held devices such as cell phones,  
3 pocket-held computer devices and the like.

4 Fig. 2 shows an exemplary computer system that can be used to implement  
5 the embodiments described herein. Computer 130 includes one or more  
6 processors or processing units 132, a system memory 134, and a bus 136 that  
7 couples various system components including the system memory 134 to  
8 processors 132. The bus 136 represents one or more of any of several types of bus  
9 structures, including a memory bus or memory controller, a peripheral bus, an  
10 accelerated graphics port, and a processor or local bus using any of a variety of  
11 bus architectures. The system memory 134 includes read only memory (ROM)  
12 138 and random access memory (RAM) 140. A basic input/output system (BIOS)  
13 142, containing the basic routines that help to transfer information between  
14 elements within computer 130, such as during start-up, is stored in ROM 138.

15 Computer 130 further includes a hard disk drive 144 for reading from and  
16 writing to a hard disk (not shown), a magnetic disk drive 146 for reading from and  
17 writing to a removable magnetic disk 148, and an optical disk drive 150 for  
18 reading from or writing to a removable optical disk 152 such as a CD ROM or  
19 other optical media. The hard disk drive 144, magnetic disk drive 146, and optical  
20 disk drive 150 are connected to the bus 136 by an SCSI interface 154 or some  
21 other appropriate interface. The drives and their associated computer-readable  
22 media provide nonvolatile storage of computer-readable instructions, data  
23 structures, program modules and other data for computer 130. Although the  
24 exemplary environment described herein employs a hard disk, a removable  
25 magnetic disk 148 and a removable optical disk 152, it should be appreciated by

A number of program modules may be stored on the hard disk 144, magnetic disk 148, optical disk 152, ROM 138, or RAM 140, including an operating system 158, one or more application programs 160, other program modules 162, and program data 164. A user may enter commands and information into computer 130 through input devices such as a keyboard 166 and a pointing device 168. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are connected to the processing unit 132 through an interface 170 that is coupled to the bus 136. A monitor 172 or other type of display device is also connected to the bus 136 via an interface, such as a video adapter 174. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

Computer 130 commonly operates in a networked environment using logical connections to one or more remote computers, such as a remote computer 176. The remote computer 176 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 130, although only a memory storage device 178 has been illustrated in Fig. 2. The logical connections depicted in Fig. 2 include a local area network (LAN) 180 and a wide area network (WAN) 182. Such networking environments are

1 commonplace in offices, enterprise-wide computer networks, intranets, and the  
2 Internet.

3 When used in a LAN networking environment, computer 130 is connected  
4 to the local network 180 through a network interface or adapter 184. When used  
5 in a WAN networking environment, computer 130 typically includes a modem 186  
6 or other means for establishing communications over the wide area network 182,  
7 such as the Internet. The modem 186, which may be internal or external, is  
8 connected to the bus 136 via a serial port interface 156. In a networked  
9 environment, program modules depicted relative to the personal computer 130, or  
10 portions thereof, may be stored in the remote memory storage device. It will be  
11 appreciated that the network connections shown are exemplary and other means of  
12 establishing a communications link between the computers may be used.

13 Generally, the data processors of computer 130 are programmed by means  
14 of instructions stored at different times in the various computer-readable storage  
15 media of the computer. Programs and operating systems are typically distributed,  
16 for example, on floppy disks or CD-ROMs. From there, they are installed or  
17 loaded into the secondary memory of a computer. At execution, they are loaded at  
18 least partially into the computer's primary electronic memory. The invention  
19 described herein includes these and other various types of computer-readable  
20 storage media when such media contain instructions or programs for implementing  
21 the steps described below in conjunction with a microprocessor or other data  
22 processor. The invention also includes the computer itself when programmed  
23 according to the methods and techniques described below.

24 For purposes of illustration, programs and other executable program  
25 components such as the operating system are illustrated herein as discrete blocks,

1 although it is recognized that such programs and components reside at various  
2 times in different storage components of the computer, and are executed by the  
3 data processor(s) of the computer.

#### 4 5 **Exemplary Implementation**

6 The inventive methods and systems discussed below are configured for use  
7 in connection with an implementation, aspects of which are described in the  
8 documents incorporated by reference above. That implementation essentially  
9 provides a single application program having a single navigable window that can  
10 be navigated to multiple different functionalities that are provided by the  
11 application program. The functionalities are extensible and can be extended via a  
12 network such as the Internet.

#### 13 14 **Use of Schema and XSL-T to Generate a User Interface**

15 When a user interacts with a DHTML document for the purpose of  
16 changing, in some way, the document through either manipulation of one or more  
17 of its values or properties, it is important that those manipulations be made, in a  
18 consistent manner, to the XML document that describes the structure of the data  
19 behind the DHTML document. In order to manipulate the XML document that  
20 describes the structure of the data behind the DHTML, there needs to be a way to  
21 transform user interactions in the DHTML to changes in the XML document. This  
22 is the problem of finding an inverse of the transformation function that is provided  
23 by the XSL-T.

24 In one implementation, the described embodiment addresses this problem  
25 by automatically (or semi-automatically, with some hint given by the application

1 developer) generating an appropriate user interface (UI) within the DHTML  
2 document that allows the user to manipulate or interact with the DHTML  
3 document. The presentation of the UI takes into account not only the XML  
4 schema, but also the XSL-T transformations that were utilized to provide the  
5 DHTML. This represents a significant departure from other XML authoring  
6 solutions that look only to the XML schema to determine what can and cannot be  
7 added to an XML document. The UIs thus allow user interaction with the  
8 DHTML view (e.g. adding and/or deleting structure) to be directly transferred  
9 back to the XML document.

10 There are many various potential types of UIs that can be presented to a  
11 user to enable them to interact with a document. Some examples include, without  
12 limitation, context blocks which are automatically added to a window based upon  
13 the user's context. Context blocks are discussed in more detail in the Application  
14 entitled "Task Sensitive Methods And Systems For Displaying Command Sets",  
15 incorporated by reference above. Other forms of UIs can include so-called  
16 widgets which are decorations within a document itself that allow a user to interact  
17 with the document. For example, if a document contains a table, there can be  
18 additional adornments around the table on which a user can click to add or delete a  
19 row or column, or to move items around within the column. Another type of UI is  
20 an accelerator which allows interaction through the keyboard. For example, if you  
21 press "Control-L" some type of predefined action is implemented.

22 In this described embodiment, a decision process is undertaken that decides  
23 which UIs to present to a user and when to present them. That is, there are  
24 potentially a number of different UI choices that can be made depending on what a  
25 user is doing in a particular document and where they are in the document. An

1 inventive approach is to utilize a number of different parameters and based upon  
2 analysis of the parameters make a decision on which UI to present to a user so that  
3 they can interact with the DHTML view. In the described embodiment, the  
4 following parameters can be used:

- 5 • Selection of where a user is in a particular DHTML document. This  
6 translates to where a user is in a particular XML document because  
7 the selection initially starts on the DHTML side and has a  
8 correspondence on the XML side;
- 9 • The portion of the XML schema that corresponds to the user's  
10 selection;
- 11 • The UI types that would be desirable to generate; and
- 12 • The XSL-T file

13 In the XSL-T file, there are certain constructs that can be suggestive of  
14 certain structures in the resultant DHTML. For example, the XSL-T file may  
15 include a "xsl:for-each" construct (i.e. for each customer, take a certain action).  
16 This construct is suggestive of a repetitive structure in the DHTML, such as a  
17 table or a paragraph. That is, if there are a number of customers, then repeating a  
18 certain action would repetitively define a certain type of structure. By considering  
19 these XSL-T constructs, certain UI types can be identified that can be displayed  
20 for the user.

21 An example is table editing. For example, if expenses are optional,  
22 according to the schema, initially there may be no expenses in a table. The XSL-T  
23 would have a "for-each" construct to render each expense, but since there are none  
24 in the XML doc, nothing is displayed. The UI should in this case produce a  
25 context block for adding an expense.

Once the first expense is created, by re-applying the XSLT transformation,  
a table is now viewable. At this point, based on the XSL-T hint that there is a



1 “for-each” associated with an expense, and the schema information that multiple  
2 expenses can be added, a decision is made to not show the “Add expense” as a  
3 context block, but to add an appropriate in-doc UI that would now take over the  
4 functionality of adding additional expenses as new rows to the table.

5 When addressing the problem of which UI to display for the user to enable  
6 interaction with a document, it is desirable to keep the overall appearance that is  
7 presented to the user as uncluttered as possible. For example, many different  
8 context blocks could be presented to user, each with its own engagable buttons that  
9 can be engaged by a user for interacting with the DHTML view. This is not  
10 desirable though because it can potentially clutter the context block area. It would  
11 be more ideal to have “in document” UIs (e.g. widget UIs) within a document that  
12 are specific to the document itself and which allow a user to interact with the  
13 document. An “in document” UI is a UI that appears within a portion of the  
14 document and enables user interaction with a portion of the document. Consider,  
15 for example, a Word document that contains an embedded drawing. When the  
16 user clicks on the embedded drawing, the drawing can appear within a frame that  
17 contains one or more buttons that can be clicked on to manipulate the drawing,  
18 e.g. a rotate button to rotate the drawing. The buttons that are associated with the  
19 embedded drawing are considered as “in document” UIs.

20 In order to provide these types of UIs, the described embodiment examines  
21 the XSL-T file to identify which UI candidates are more suited to have their  
22 functionalities provided by “in document” UIs.

23 For example, if the schema specifies that multiple expenses are allowed,  
24 and the XSL-T has a “for-each” construct for expenses, by looking at the first  
25 element introduced by the XSL-T after an expense is matched, it could determine

1 what kind of helpful UI to add. If an DHTML TABLE is created, then it should be  
2 adorned with table-editing widgets, but if there is SPAN, for example, then create  
3 a context block, and not an in-doc UI.

4 That is, the above-described context blocks are not “in document” because  
5 they are provided within a pane that is disposed adjacent a document area within  
6 which a user can work on a document. One goal of the described embodiment is  
7 to identify UIs, based upon the analysis discussed below, that are the best  
8 candidates for incorporation as “in document” UIs that specifically adorn  
9 document portions and permit user interaction with the document itself.

10 Consider that, without taking into account the XSL-T in the analysis of  
11 which UIs to present to a user, the only UIs that could be presented would not be  
12 in-document UIs. The context blocks are the most generic UI constructs in the  
13 present example. But if we know that we have a table created in DHTML, then  
14 the context blocks can be replaced by in-doc constructs. By inspecting the XSL-T  
15 we can find out what DHTML construct is created by the XSLT transformation.  
16 That is, without consideration of the XSL-T, only generic UIs, e.g. the context  
17 block UIs, would likely be generated. For example, if a user is working within a  
18 DHTML document that contains a table, a context block can be provided that  
19 enables the user, through manipulation of various “out of document” UIs to  
20 manipulate the table, e.g. adding a row, column and the like. By considering the  
21 XSL-T, the UI that is produced can be refined and the context blocks, or at least a  
22 portion of the functionality that is provided by the context blocks, can be replaced  
23 with other types of in document UIs . The XSL-T is thus used for refinement of  
24 the UIs.  
25

Fig. 3 shows a flow diagram that describes steps in a UI generation method in accordance with this described embodiment. Step 300 makes a selection in a DHTML document. This step is implemented by a user moving their cursor to a particular area within a document. Step 302 determines, based upon the user's selection, the corresponding selection in the XML document. For example, if a user has selected a particular portion of a table used to display a specific fragment of the XML document, then this step determines the exact fragment of the XML document that corresponds to the user's selection. Based on the selection in the XML document, step 304 determines the appropriate place in the XML schema that corresponds to the selection and the various types of actions that can be taken from this selection. The various types of actions correspond to the various ways in which a user might manipulate the portion of the document that they have selected. Step 306 then produces the appropriate operations that can be undertaken for the various action types. For example, if the user is working in a table, this step might produce operations for adding a row or column or deleting a row or column. Once the operations are produced by step 306, step 308 determines, from the XSL-T file, what type of UI to display for the user. If the XSL-T is not considered in this process, then the available UIs would be presented as context blocks (i.e. not "in document" UIs). By using the XSL-T, the described embodiment refines the production of context blocks by reducing the number of context blocks that are produced and, instead, producing "in document" UIs that now relocate the functionality that would otherwise be provided by the context blocks.

## Manipulation of XML Structures Using Crystals

Recall that one of the benefits of XML is the richness with which data can be described. XML, by its very nature, can provide a wide variety of variations of data. Because of this, UI solutions for interacting with data (displayed in DHTML using XSL-T) have been hardcoded and specific to individual schemas. This is a manifestation of the ease with which hardcoded solutions can be provided through XSL-T.

In one described embodiment, the notion of a crystal is introduced to enable interactions with a DHTML view to be directly mapped back to the XML file or tree. Advantageously, the crystals are configured to work on various data shapes, independent of the XML schemas. This means that when the data has a particular shape, as defined by the XML tree that contains the data, specific crystals that are configured for that particular shape can be used to render the DHTML and also ensure that user interactions with the DHTML view are directly mapped back to the XML tree. The crystals do not care about the specific data that is provided by the data shape, nor the schema or tags that are used to contain the data.

Consider, for example, Fig. 4 which shows an XML document 400, a crystal 402 and the resultant DHTML document 404. In one basic form, a crystal comprises one or more behaviors 406 and the basic XSL-T 408 that is utilized to transform the XML into the DHTML. The behaviors are implemented, in this particular example, as binary code that is associated with or attached to the DHTML tags that are generated by the XSL-T. Consider, for example, the hierarchical tree that is shown directly below XML document 400. This hierarchical tree represents a portion of an XML tree that is maintained in memory. In this example, the tree has a “products” root node and a “product 1”

This is diagrammatically illustrated in Fig. 4 by the DHTML tree structure shown underneath the DHTML view 404. There, a node corresponding to the “product” node is shown adorned with a behavior. This behavior is binary code that enables a user to interact, via an appropriate UI (such as an in document “add product” button 411 attached to the table) with the DHTML view and have any defined modifications made by the user mapped back to the appropriate XML tree. When a user interacts with the DHTML view, the XML tree is structurally

1 manipulated (as by adding the appropriate tags and structure), and then the XSL-T  
2 is invoked to redisplay the DHTML view.

3 In the purchase order example, assume that the user adds a new product to  
4 the DHTML view table by clicking on "add product" button 411 which adds a new  
5 row to table 410. In this example, when the new product is added, the behavior or  
6 binary code maps the modification back to the XML tree and incorporates the  
7 modification by making a structural change to the XML tree. In this specific  
8 example, the structural change would include adding a branch to the XML tree to  
9 represent the newly-added product. This added branch is shown as the dashed  
10 branch on the "Products" XML tree.

11 Consider the second XML tree 412 shown directly below the Products  
12 XML tree. That tree is an "Addresses" XML tree and is associated with addresses  
13 that might appear in an address book. This data is extremely different from the  
14 data that is associated with the Products XML tree. In fact, there is no relation at  
15 all between the data. Notice, however, that the Addresses XML tree has the same  
16 *shape* as the XML tree appearing directly above it. In the described embodiment,  
17 a similar crystal can be used to render a DHTML address book that contains  
18 entries for a name, street and zip code. The crystal would likely contain slightly  
19 different XSL-T for labeling purposes, but can contain the same exact behavior  
20 that was utilized in the above example to manipulate the structure of the Products  
21 XML tree. To this extent, a user interface button 411 is provided on the Address  
22 table and includes the same behavior as the user interface button associated with  
23 the Products table. Thus, when a user adds an entry to their address book, the  
24 behavior, or binary code, that is associated with the DHTML "Address" tag would  
25

1 ensure that any changes made to the DHTML view are mapped directly back to  
2 the corresponding XML document.

3 The crystals can advantageously be prepackaged software containers that  
4 include the behaviors that are specific to the shape of the data and not necessarily  
5 dependent upon the schema or specific data that may be contained by an XML  
6 document. This approach is very well suited to handling complex XSLT  
7 transformations which naturally flow from the robustness that XSL-T provides.  
8 By incorporating and associating behaviors in the DHTML tree, problems  
9 associated with handling complex XSLT transformations insofar as XML  
10 authoring is concerned are solved. This approach is extremely flexible and is not  
11 tied to any one schema or specific data, as were the solutions in the past. This  
12 approach also provides the application developer with the ability to develop  
13 complex XSL-T, without worrying about how the underlying XML is going to be  
14 manipulated responsive to a user manipulating the DHTML document. Further,  
15 because the approach utilizes crystals having behaviors that are specific to data  
16 shape and not specific to schema or data, the crystals are reusable across any XML  
17 documents that have shapes that correspond to the shapes for which the various  
18 crystals were designed.

19 Fig. 5 is a flow diagram that shows steps in a method in accordance with  
20 the embodiment described above. Step 500 defines multiple crystals each of  
21 which include one or more behaviors. In the described example, behaviors are  
22 implemented as binary code. The behaviors are specific to a data shape and do not  
23 depend on a schema or specific data. Step 502 uses one or more of the crystals to  
24 render a DHTML view from an XML document. Step 504 attaches at least one  
25 behavior to a DHTML tag. The behavior ensures that any modifications that are

made to the DHTML view are mapped back to and appropriately change the XML document that contains the data in the DHTML view. Step 506 interacts with the DHTML view in some way, based upon user input via a UI. This step can be implemented by a user interacting with some type of structure, for example a table, within the DHTML view. Responsive to the user interaction with the DHTML view, step 508 uses the behavior to map the user's interaction back to the XML document and make the appropriate structural changes in the XML tree that contains the data. For example, the XML branch in Fig. 4 off of the "Products" node, indicated with a dashed line, might be the result of a user who adds a new product to the purchase order provided in the DHTML view.

### **Example**

The above approach is very flexible and can be conveniently used by application developers to provide applications. Assume that an independent software vendor (ISV) develops applications for end users and he wants to construct a purchase order. The ISV can select an appropriate XML schema for the purchase order which would then define the types of tags that the purchase order can contain. The ISV would need to write the appropriate XSL-T that could present the purchase order in DHTML in a ISV-defined manner. Perhaps the ISV wants to make the purchase order specific to a particular company. The XSL-T provides a way for the ISV to do this. That is, each ISV may wish to present their data differently in a way that is specific to the ISV. Thus, while they each may use the same schema, there will be many different instances of the schema each of which can be potentially very different from the others. One goal of the crystal-based implementation discussed above, is that it should be very easy for ISVs to



1 develop applications based on XML. Accordingly, when the ISV writes their  
2 XSL-T, they can incorporate various behaviors that are provided by multiple  
3 different crystals. These crystals are predefined so that the ISV need not worry  
4 about defining them. They can simply select the crystals that are appropriate for  
5 their shape of data, and incorporate them with XSL-T. Now, when the XML is  
6 transformed into DHTML, user interactions with the DHTML view can be  
7 mapped to the underlying XML document.

8 Although the invention has been described in language specific to structural  
9 features and/or methodological steps, it is to be understood that the invention  
10 defined in the appended claims is not necessarily limited to the specific features or  
11 steps described. Rather, the specific features and steps are disclosed as preferred  
12 forms of implementing the claimed invention.